Mesurer les performances (CPU) sous Linux

Résumé :

Il existe des outils permettant de tracer les temps CPU utilisés (*profiling*) dans *Code_Aster*.

Sur Linux, on utilise l'outil `gprof`. L'usage de cet outil impose de compiler tous les sources avec l'option "`-pg`". Le sur-coût de l'instrumentation est négligeable. La résultat du *profiling* est un fichier texte qu'il faut interpréter.

Pour simplifier l'interprétation, on propose en fin de document un outil pour tracer un graphe à partir du fichier texte produit par le calcul « profilé ».

Copyright 2015 EDF R&D - Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Table des matières

1 Gprof....3
1.1 Pour aller vite....3
1.2 Si l'on a besoin de "surcharger" le code....4
1.3 Instrumentation d'un exécutable avec f77 -pg (ou cc -pg)....4
1.4 Sur-coût de l'instrumentation....5
1.5 Exécution du Code instrumenté avec Astk....5
1.6 Exploitation des résultats....6
1.7 Dépouiller les résultats du profiling....6
1.8 Analyse de l'exemple....8

Copyright 2015 EDF R&D - Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

1 Gprof

1.1 Pour aller vite

Si l'on a la chance de vouloir "profiler" la version officielle sur la machine `clpaster` (ou la machine `bull`), la manière la plus simple de le faire consiste à remplacer l'exécutable (bouton SURCHARGE) par le petit script ci-dessous. On récupère alors à la fin du fichier message le listing de gprof.

Remarques :

Attention : l'exécution de l'outil gprof est très couteuse en temps `CPU` . Par exemple, sur la machine `clpaster` , pour une exécution Aster de 5s, gprof consomme 3 minutes. Sur la machine bull , c'est encore pire (25 minutes !). Mais on peut penser que ce sur-coût dépend peu de la durée de l'exécution profilée (c'est une constante).

Cette technique fonctionne également avec as_run

fichier astk_perf.sh :

```
#!/bin/bash
#
# Encapsulation de l'execution du Code_Aster pour pouvoir mesurer
# les performances CPU avec gprof:
#-------------------------------------------------------------------------

#  Mode d'emploi :
#  ================
#  1) recopier ce script pour pouvoir éventuellement le modifier:
#
#  2) le vérifier : (choix de l'exécutable profilé)
#
#  3) lancer astk (ou "run_aster")
#     et mettre comme "excutable" ce script (il peut rester sur la machine
locale)
#
#  4) le resultat des mesures sera imprimé dans le fichier fort.6 (.mess)
#-------------------------------------------------------------------------

# 1. choix de l'executable instrumenté Aster :
#---------------------------------------------
EXECUT=/home/aster/NEW10/asterp   # sur clpaster
#EXECUT=/aster/NEW10/asterp        # sur bull
echo "exécutable utilisé: "$EXECUT

# 3. on lance l'exécutable instrumenté (avec tous ses arguments) :
#-----------------------------------------------------------------
$EXECUT  $*   >> fort.6
STATUS=$?
echo " fin exécution " >>fort.6
ls -ltr >> fort.6

# 4. on lance gprof pour écrire au bout de fort.6 (fichier message) :
#--------------------------------------------------------------------
gprof   $EXECUT  gmon.out  >> fort.6
echo " fin écriture listing de gprof " >>fort.6

exit $STATUS
```


Copyright 2015 EDF R&D - Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Copyright 2015 EDF R&D - Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Attention :

il ne faut pas oublier de donner les droits d'exécution à ce script :

```
[desoza@aster2 PROF]$ chmod a+x astk_perf.sh
[desoza@aster2 PROF]$ ls -l astk_perf.sh
-rwxr-xr-x  1 desoza astergrp 1320 Aug 19 18:29 astk_perf.sh
```

1.2 Si l'on a besoin de "surcharger" le code

Sur certaines machines, des bibliothèques profilées (et les exécutables correspondants) sont mis à jour chaque semaine :

Machine Bull /aster/NEW10 :

```
fichier de configuration : ./config.prof
exécutable               : ./asterp
bibliothèques profilées  : ./prof/lib/*
```

Machine clpaster /home/aster/NEW10 :

```
fichier de configuration : ./config.profiling
exécutable               : ./asterp
bibliothèques profilées  : ./lib_p/*
```

Sur ces deux machines, il est possible de surcharger la version profilée (pratique lorsque l'on souhaite tester un nouveau développement touchant aux performances). Pour cela il faut ajouter un champ supplémentaire dans l'onglet surcharge d'`astk` et mettre en donnée (`D`) le fichier de configuration config.prof (`type "conf"`).

1.3 Instrumentation d'un exécutable avec `f77 -pg` (ou `cc -pg`)

Lorsqu'on utilise une machine (ou une version du code) pour laquelle on ne dispose pas des bibliothèques profilées, il faut recompiler tous les sources.

Pour réaliser cette recompilation avec `Astk`, il faut :

- "surcharger" tous les sources (`F77` et `C`). Pour gagner du temps, on peut concaténer les sources `F77` en "paquets" (300 routines par exemple).
- modifier le fichier `"config.txt"` pour ajouter l'option `"-pg"` sur les 5 lignes suivantes :

```
OPTL   | f90  | ?   | -v  -pg
OPTC_D | cc   | ?   | -c -g -pg  -DP_LINUX
OPTC_O | cc   | ?   | -c    -pg  -DP_LINUX
OPTF_D | f90  | ?   | -c -g  -pg -I/opt/mpich2-1.0.1/include
OPTF_O | f90  | ?   | -c -O2 -pg -I/opt/mpich2-1.0.1/include
```

Remarques :

Le fichier `config.txt` ainsi modifié permet d'instrumenter le code en mode `"debug"` et `"nodebug"` . Le mode `"nodebug"` est a priori préférable pour mesurer les "vraies" performances du code. En revanche, le mode `"debug"` est nécessaire si l'on veut connaître les lignes les plus consommatrices.
J'ai malheureusement observé un problème inexplicable en mode `"debug"` : le résultat du `profiling` indiquait des liens d'appel entre routines qui n'existaient pas ! On peut toutefois espérer que cette anomalie n'invalide pas entièrement le reste de la mesure.

Copyright 2015 EDF R&D - Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

1.4 Sur-coût de l'instrumentation

A titre d'exemple, j'ai profilé le test ssnv506c et j'ai obtenu les résultats globaux suivants :

* en mode nodebug sans instrumentation	138s
* en mode nodebug avec instrumentation	139s
* en mode debug sans instrumentation	218s
* en mode debug avec instrumentation	228s

On constate que l'instrumentation a un coût CPU négligeable.

1.5 Exécution du Code instrumenté avec Astk

Une fois cette instrumentation faite, il faut exécuter l'étude que l'on veut "profiler" avec l'exécutable que l'on vient de produire. L'exécution de l'étude produit un fichier (appelé gmon.out) dans le répertoire temporaire d'exécution.

Pour conserver le précieux fichier gmon.out, trois possibilités existent :

- La première est d'utiliser Astk en interactif et cliquer le bouton "lancer pre" (au lieu du classique "lancer run"). Cette option d'Astk permet de préparer l'environnement d'exécution. On se place ensuite dans le répertoire préparé et on "lance" Aster manuellement. Il s'agit de la même "astuce" que pour l'utilisation d'un débogueur.

- La deuxième possibilité consiste à rajouter un champ dans le profil astk de type "nom" et dont la valeur est "./gmon.out". On récupérera alors dans le répertoire correspondant au chemin de base, un fichier de nom gmon.out. Attention : ceci ne fonctionne qu'en interactif (*i.e.* pas en batch sur clpaster et Bull).

- La troisième est de modifier le script proposé au-dessus en remplaçant

```
# 4. on lance gprof pour écrire au bout de fort.6 (fichier message) :
#-------------------------------------------------------------------
gprof   $EXECUT  gmon.out  >> fort.6
echo " fin écriture listing de gprof " >> fort.6
```

par

```
# 4. on lance gprof pour écrire au bout de fort.6 (fichier message) :
#-------------------------------------------------------------------
mv gmon.out ./REPE_OUT/.
gprof   $EXECUT  ./REPE_OUT/gmon.out   >> fort.6
echo " fin écriture listing de gprof " >> fort.6
```

et de définir dans Astk un "repe" en résultat. Dans ce répertoire on retrouve le fichier "gmon.out", et cela aussi bien en interactif qu'en batch.

Copyright 2015 EDF R&D - Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Ainsi une méthodologie de profiling (par exemple sur bull) est la suivante:

- Création d'un profil d'étude astk
  - on renseigne les fichiers de l'étude fournie, et un champ de type "nom" dont la valeur est "./gmon.out" ou un répertoire dans lequel le fichier "./gmon.out" sera copié.

- Ajout d'une surcharge:
  - on renseigne les fichiers de surcharge : types "f", "c", ...
  - on renseigne le fichier de configuration (type "conf") "config.prof" localisé dans /aster/v10/NEW10
  - on crée un exécutable "perso" (par exemple : /home/user/asterp)

- Exécution du profil d'étude par astk

- Création du listing
  - on se place dans le répertoire où a été généré le fichier gmon.out, puis on lance l'instruction suivante: gprof /home/user/asterp gmon.out > listing

- Edition du listing pour pouvoir commencer le dépouillement.

1.6 Exploitation des résultats

Une fois l'étude exécutée et le fichier "gmon.out" récupéré, on peut analyser ce fichier avec la commande :

```
gprof mon_executable gmon.out > listing
```

Attention :

> *il ne faut pas se décourager. Pour une exécution de 30s, on a déjà vu, la commande gprof consommer plus de 5 minutes de CPU. Le temps de gprof ne dépend pas trop (a priori) du temps de l'exécution profilée.*

L'interprétation du fichier obtenu (listing) est décrite ci-dessous. Un excellent document décrivant tout le processus de profiling est celui écrit par Jay Fenlason et Richard Stallman : "Gnu gprof The GNU profiler". On le trouve facilement sur le Web.

Remarque :

> *Même si l'on recompile tous les sources d'Aster, la "profondeur" de l'analyse des performances s'arrêtent aux bibliothèques que l'on utilise à l'édition des liens et qui n'ont pas été compilées avec "-pg" . C'est par exemple le cas des routines BLAS . Le temps consommé dans ces bibliothèques ne peut pas être rattaché aux routines d'Aster qui les appellent. Ce défaut peut être important, par exemple, si on veut mesurer les performances des solveurs MUMPS ou MULT_FRONT car une grande partie du temps consommé l'est dans des routines BLAS .*

1.7 Dépouiller les résultats du profiling

Par défaut, le fichier est lourd. Il est possible de limiter l'affichage des infos en jouant avec les options de gprof. Les "temps systèmes" sont indiqués sous forme de nombre d'instructions utilisées.
On va détailler un peu, en commençant par la fin du fichier :

```
Index by function name
 [401] PyArg_Parse            [591] cftabl_    [1000] proc_at_0x1213acb50
 [212] PyArg_ParseTuple       [84] cftyli_     [660]  proc_at_0x1213ad470
[1137] PyArg_ParseTupleAnd    [310] cgmacy_    [453]  proc_at_0x1213ad560
[1605] PyBuffer_FromObject    [79] charme_     [680]  proc_at_0x1213aeac0
```


Copyright 2015 EDF R&D - Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

```
[1256] PyCFunction_Fini       [476] chlici_   [1221] proc_at_0x1213aedc0
 [531] PyCFunction_New        [190] chloet_   [217]  proc_at_0x1213b18e0
[1549] PyCObject_AsVoidPtr    [226] chmano_   [629] proc_at_0x1213b1e00Y
```

Chaque fonction appelée lors de l'exécution est repérée par un numéro entre crochet.
Juste au dessus :

```
granularity: instructions; units: inst's; total: 201924201580.70 inst's

 <A>  <B>            <C>           <D>  <E>        <F>        <G>
49.6  100384307222  100384307222  161  623505013  623596299 tldlr8_ [16]
31.0   63144941823  62760634601   506  124032874  124101882 rldlr8_ [17]
```

Ce tableau résume les appels les plus fréquents.

- COLONNE <A> : pourcentage du nombre d'instructions exécutées par cette fonction par rapport au total de l'exécution.
- COLONNE <B> : nombre d'instructions cumulées par cette fonction et celles qui précèdent.
- COLONNE <C> : nombre d'instructions pour cette fonction.
- COLONNE <D> : nombre d'appels a cette fonction
- COLONNE <E> : rapport entre la colonne <B> et la colonne <D> (nombre d'instructions moyen par appel de la fonction)
- COLONNE <F> : nombre moyen d'instructions par appel de la fonction et de ses descendants.
- COLONNE <G> : nom de la fonction et son numéro de référence (entre crochets).

Dans cet exemple, la fonction `tldlr8` a pris 49.4% du total du calcul en étant appelée 161 fois.

Enfin, au début du fichier, nous avons l'arbre d'appel complet. Il sera trié par ordre d'appel (on commence par le main et on descend) ou par une fonction (voir les options de `gprof`).

Prenons l'exemple de `tldlr8` :

```
<A>     <B>  <C>              <D>          <E>          <F>

             100263313681.76  14679301.29  161/161      tldlgg_ [15]
[16]    49.7 100263313681.76  14679301.29  161          tldlr8_ [16]
             3129121.03       6207534.02   4485/30537   __upcUpcall [352]
             35974.59         2749927.50   522/195235   jelibe_ [65]
             192341.36        1770419.18   1005/775659  jeveuo_ [56]
             47302.73         140745.02    161/202579   jedema_ [102]
             18938.92         126525.05    322/63148    jeexin_ [196]
             27722.26         85430.33     94/49118     jeecra_ [154]
             17033.41         67779.29     94/13206     jecreo_ [257]
             45068.75         84.88        1044/1075446 jexnum_ [163]
             13618.68         2023.63      161/202581   jemarq_ [205]
             1710.66          0.00         161/3481     infniv_ [853]
```

On repère l'instruction de l'arbre d'appel par le numéro entre crochets à gauche. Ici, le numéro [16] indique la fonction `tldlr8_` (comme indiqué à la fin du fichier par exemple). C'est la fonction-référence (le nœud de l'arbre). Les lignes au dessus sont les appelants de cette fonction (ce sont les fonctions-parents), ceux en dessous sont les fonctions appelées (ce sont les fonctions-enfants). Chaque fonction a deux chiffres principaux : le nombre d'instructions exécutées dans elle-même (instruction " terminale " du `FORTRAN`) et le nombre d'instructions exécutées dans les fonctions-enfants.

```
Fonction-parent
Fonction-parent
...
Fonction-référence
```


Copyright 2015 EDF R&D - Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

```
      Fonction-enfant
      Fonction-enfant
      ...
```


Copyright 2015 EDF R&D - Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Pour la fonction-référence :

- COLONNE <A> : numéro de repérage de la fonction-référence.
- COLONNE <B> : le chiffre 49.7 est le pourcentage du nombre d'instructions exécutées par cette fonction-référence par rapport au total de l'exécution (idem tableau précédent)
- COLONNE <C> : nombre d'instructions pour la fonction-référence elle-même.
- COLONNE <D> : nombre d'instructions pour les fonctions-enfants de la fonction-référence.
- COLONNE <E> : nombre de fois ou la fonction a été appelée
- COLONNE <F> : nom de la fonction-référence

Pour les fonctions-parents et les fonctions-enfants :

- COLONNE <A> : vide
- COLONNE <B> : vide
- COLONNE <C> : nombre d'instructions pour la fonction elle-même.
- COLONNE <D> : nombre d'instructions pour les descendants de la fonction
- COLONNE <E> : donne deux chiffres a/b dont le sens varie suivant le type de fonction (parent ou enfant par rapport à la fonction référence) :
  - Pour les fonctions-parents (au-dessus de la fonction référence) a/b : <a> est le nombre de fois où la fonction-référence a été appelée par cette fonction-parent par rapport au nombre total <b> d'appels de la fonction-référence.
  - Pour les fonctions-enfants (en-dessous de la fonction référence) a/b : <a> est le nombre de fois où la fonction-enfant a été appelée par la fonction-référence par rapport au nombre total <b> d'appels de la fonction-enfant.
- COLONNE <F> : nom de la fonction

Remarques :

Si le nombre d'instructions pour les descendants d'une fonction vaut zéro, c'est que la fonction considérée n'en appelle aucune autre. On est " au bout " de l'arbre, il n'y a que des appels FORTRAN de base dans la fonction. (c'est le cas de `infniv` par exemple)
Pour une fonction-référence donnée, si on fait la somme des <a> dans la colonne <E> des fonctions parents, on obtient le nombre d'appels total de la fonction référence.
Pour une fonction-référence donnée, si on fait la somme des colonnes <C> et <D> de ses fonctions-enfants, on obtient le chiffre de la colonne <D> de la fonction-référence.

1.8 Analyse de l'exemple

Dans l'exemple présenté, la fonction tldlr8 est coûteuse puisqu'à elle-seule, elle représente près de la moitié du nombre d'instructions total de l'exécution. On voit également que ce sont ses propres instructions qui prennent du temps et non l'appel à ses fonctions-enfants (le rapport entre les deux atteint 1000). Comme seule la fonction `tldlgg` appelle `tldlr8`, il faut regarder l'arbre d'appel pour cette fonction. On voit alors que c'est l'algorithme de contact/frottement (`fropgd`) qui est le plus glouton (les 2/3 des appels à `tldlgg` sont faits par l'algorithme de contact).

1.9 Génération d'un graphe avec *gprof2dot*

1.9.1 Présentation

Pour faciliter l'interprétation d'un *profiling*, on décrit dans cette section un petit utilitaire Python (*gprof2dot*) qui transforme le fichier texte produit par `gprof` en un graphe plus simple à lire.

Copyright 2015 EDF R&D - Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Le graphe produit est celui des routines parcourues avec report des colonnes <B> et <C> (en % du nombre total d'instructions), du nombre d'appels de la routine. Par ailleurs les cellules du graphe sont coloriées (du bleu vers le rouge) pour identifier très vite les chemins critiques.
On trouvera plus d'informations sur la page web du développeur de *gprof2dot* : http://code.google.com/p/jrfonseca/wiki/Gprof2Dot. Le script python est directement accessible ici.

1.9.2 Utilisation

Une fois le fichier d'*output* produit par `gprof` récupéré, celui-ci s'appelant gprof.output, on exécutera la commande suivante dans un terminal :

desoza@claut621 :/tmp$ cat /chemin/vers/gprof.output | /chemin/vers/gprof2dot.py | dot -Tpng -o graphe.png

Le fichier PNG produit pourra alors être ouvert avec tout visionneur d'images. Un exemple du type de fichier produit est donnée dans la figure suivante :

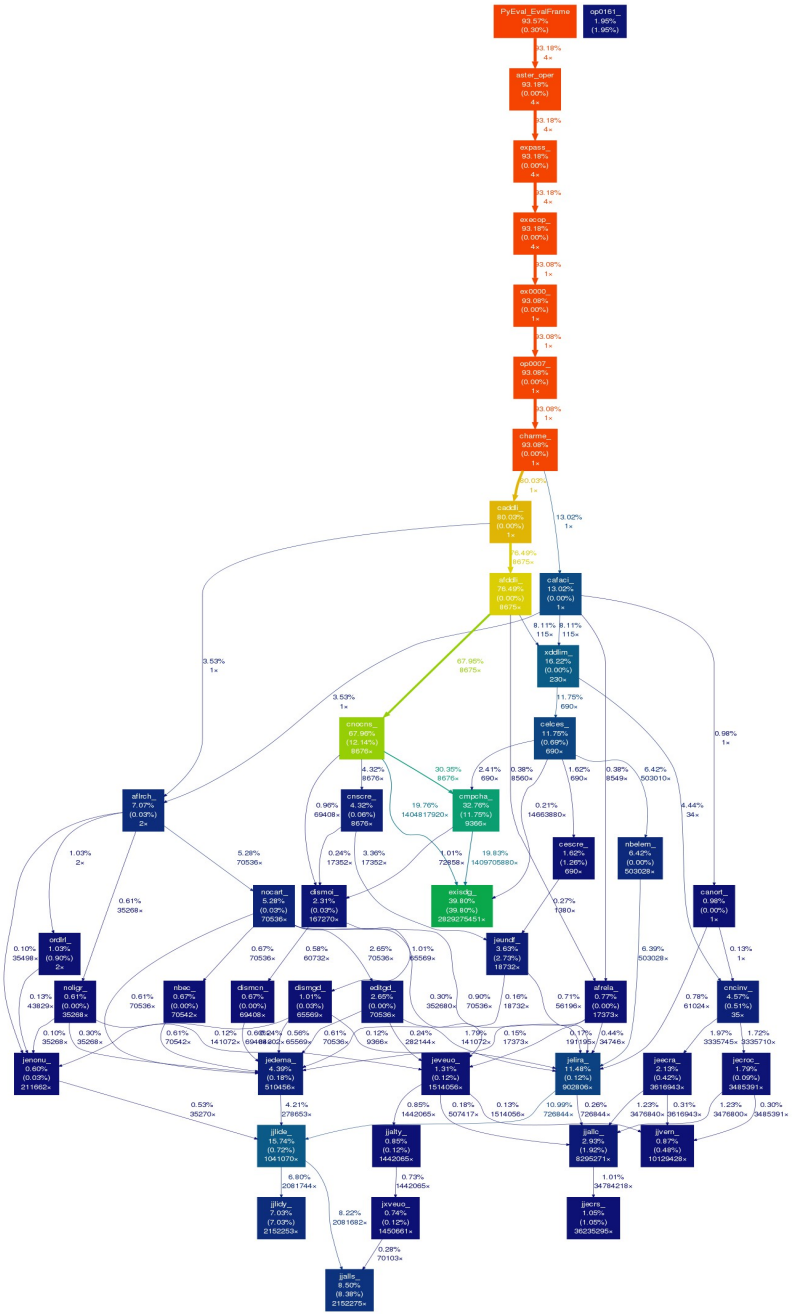

Figure 1 : graphe d'appel généré avec *gprof2dot*

Copyright 2015 EDF R&D - Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)